Tobacco use and mortality, 2004-2015

1. Abstract

This project aims to analyze historical tobacco use data and its impact on mortality across various U.S. regions from 2004 to 2015. Using machine learning techniques, particularly linear regression, we built a predictive model to estimate tobacco-related deaths based on factors such as smoking rate, prescription frequency, price index, and affordability. Finally, the model is integrated into a Flask-based web application for user-friendly prediction.

2. Introduction

2.1 Problem Background

Tobacco use remains one of the leading causes of preventable deaths worldwide. Governments and health organizations track various metrics to assess and reduce smoking-related harm, including affordability, taxation, and medical interventions.

2.2 Project Purpose

This project is intended to:

- Analyze data related to tobacco usage and health outcomes.
- Model mortality outcomes using machine learning.
- Deploy a user interface where users can enter tobacco indicators and get predicted mortality rates.

3. Objectives

- Understand the trends and patterns in tobacco use and associated deaths.
- Clean and preprocess multiple datasets into a unified format.
- Train a predictive model using machine learning (Linear Regression).
- Create a web application using Flask for real-time prediction.

4. Dataset Description

The project combines five datasets, each contributing unique insights:

- **smokers.csv**:
  - Historical smoking prevalence from 1974 onward, categorized by age groups.
  - Needs filtering for years 2004–2015.
- **prescriptions.csv**:
  - Pharmacotherapy prescriptions (NRT, Zyban, Champix) from **2010/11 to 2014/15**.
  - Data includes prescription counts and net cost.
- **metrics.csv**:
  - Economic and affordability indices, including tobacco price index, disposable income, and expenditure on tobacco.
- **fatalities.csv**:
  - Mortality data by ICD10 disease categories.

- Includes total deaths and those attributable to smoking (e.g., cancer, respiratory, circulatory).

- **admissions.csv**:
  - Healthcare admissions data (structured similar to fatalities.csv).

5. Code Explanation

5.1 Install the necessary libraries:

```
pip install pandas numpy matplotlib seaborn scikit-learn flask
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: flask in /usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: Werkzeug>=3.1 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.6)
Requirement already satisfied: itsdangerous>=2.2 in /usr/local/lib/python3.11/dist-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from flask) (8.1.8)
Requirement already satisfied: blinker>=1.9 in /usr/local/lib/python3.11/dist-packages (from flask) (1.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1.2->flask) (3.0.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

5.2 Import Libraries

```
04] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import r2_score
    import joblib
```

- This code imports key libraries for data analysis and machine learning:
  - **pandas** for data manipulation.
  - **numpy** for numerical operations.
  - **matplotlib** and **seaborn** for visualizations.
  - **scikit-learn** for splitting data, training a linear regression model, and evaluating it.
  - **joblib** for saving trained models.

5.3 Load the Data

```python
# Load CSV files
smokers_df = pd.read_csv('/content/smokers.csv')
prescriptions_df = pd.read_csv('/content/prescriptions.csv')
metrics_df = pd.read_csv('/content/metrics.csv')
fatalities_df = pd.read_csv('/content/fatalities.csv')
admissions_df = pd.read_csv('/content/admissions.csv')
```

```python
smokers_df.head()
```

	Year	Method	Sex	16 and Over	16-24	25-34	35-49	50-59	60 and Over
0	1974	Unweighted	NaN	46	44	51	52	50	33
1	1976	Unweighted	NaN	42	42	45	48	48	30
2	1978	Unweighted	NaN	40	39	45	45	45	30
3	1980	Unweighted	NaN	39	37	46	44	45	29
4	1982	Unweighted	NaN	35	35	38	39	41	27

```python
prescriptions_df.head()
```

	Year	All Pharmacotherapy Prescriptions	Nicotine Replacement Therapy (NRT) Prescriptions	Bupropion (Zyban) Prescriptions	Varenicline (Champix) Prescriptions	Net Ingredient Cost of All Pharmacotherapies	Net Ingredient Cost of Nicotine Replacement Therapies (NRT)	Net Ingredient Cost of Bupropion (Zyban)	Net Ingredient Cost of Varenicline (Champix)
0	2014/15	1348	766	21	561.0	38145	18208	807	19129.0
1	2013/14	1778	1059	22	697.0	48767	24257	865	23646.0
2	2012/13	2203	1318	26	859.0	58121	28069	994	29058.0
3	2011/12	2532	1545	30	957.0	64552	30951	1216	32385.0
4	2010/11	2564	1541	36	987.0	65883	30808	1581	33494.0

```python
metrics_df.head()
```

	Year	Tobacco Price\nIndex	Retail Prices\nIndex	Tobacco Price Index Relative to Retail Price Index	Real Households' Disposable Income	Affordability of Tobacco Index	Household Expenditure on Tobacco	Household Expenditure Total	Expenditure on Tobacco as a Percentage of Expenditure
0	2015	1294.3	386.7	334.7	196.4	58.7	19252.0	1152387.0	1.7
1	2014	1226.0	383.0	320.1	190.0	59.4	19411.0	1118992.0	1.7
2	2013	1139.3	374.2	304.5	190.3	62.5	18683.0	1073106.0	1.7
3	2012	1057.8	363.1	291.3	192.9	66.2	18702.0	1029378.0	1.8
4	2011	974.9	351.9	277.1	189.3	68.3	18217.0	990828.0	1.8

```python
fatalities_df.head()
```

	Year	ICD10 Code	ICD10 Diagnosis	Diagnosis Type	Metric	Sex	Value
0	2014	All codes	All deaths	All deaths	Number of observed deaths	NaN	459087
1	2014	C33-C34 & C00-C14 & C15 & C32 & C53 & C67 & C6...	All deaths which can be caused by smoking	All deaths which can be caused by smoking	Number of observed deaths	NaN	235820
2	2014	C00-D48	All cancers	All cancers	Number of observed deaths	NaN	136312
3	2014	J00-J99	All respiratory diseases	All respiratory diseases	Number of observed deaths	NaN	61744
4	2014	I00-I99	All circulatory diseases	All circulatory diseases	Number of observed deaths	NaN	126101

```
[ ] admissions_df.head()
```

	Year	ICD10 Code	ICD10 Diagnosis	Diagnosis Type	Metric	Sex	Value
0	2014/15	All codes	All admissions	All admissions	Number of admissions	NaN	11011882
1	2014/15	C33-C34 & C00-C14 & C15 & C32 & C53 & C67 & C6...	All diseases which can be caused by smoking	All diseases which can be caused by smoking	Number of admissions	NaN	1713330
2	2014/15	C00-D48	All cancers	All cancers	Number of admissions	NaN	1691035
3	2014/15	J00-J99	All respiratory diseases	All respiratory diseases	Number of admissions	NaN	611002
4	2014/15	I00-I99	All circulatory diseases	All circulatory diseases	Number of admissions	NaN	907157

5.4 Data Cleaning

```
[231] def clean_columns(df):
          df.columns = df.columns.str.replace(r'\n', ' ', regex=True).str.strip()
          return df

      # Apply column cleaning
      smokers_df = clean_columns(smokers_df)
      prescriptions_df = clean_columns(prescriptions_df)
      metrics_df = clean_columns(metrics_df)
      fatalities_df = clean_columns(fatalities_df)
      admissions_df = clean_columns(admissions_df)
```

This Code Explains:

```
df.columns = df.columns.str.replace(r'\n', ' ', regex=True).str.strip()
```

- **df.columns** accesses the column names of the DataFrame.
- **.str.replace(r'\n', ' ', regex=True)** replaces any newline character (\n) with a space using a regular expression.
- **For example:** "Smoking\nRate" → "Smoking Rate"
- **.str.strip()**
  - Removes any leading or trailing whitespace from the column names.
  - For example:
    " Death Rate " → "Death Rate"
- Applies the clean_columns() function to five DataFrames.

5.5 Filter Years (2004–2015) and Format

```
[232] # Fix Year columns
      # Extract and convert Year where necessary
      prescriptions_df['Year'] = prescriptions_df['Year'].str[:4].astype(int)
      admissions_df['Year'] = admissions_df['Year'].str[:4].astype(int)
      fatalities_df['Year'] = fatalities_df['Year'].astype(int)

      # Filter years
      smokers_df = smokers_df[(smokers_df['Year'] >= 2004) & (smokers_df['Year'] <= 2015)]
      prescriptions_df = prescriptions_df[(prescriptions_df['Year'] >= 2004) & (prescriptions_df['Year'] <= 2015)]
      metrics_df = metrics_df[(metrics_df['Year'] >= 2004) & (metrics_df['Year'] <= 2015)]
      fatalities_df = fatalities_df[(fatalities_df['Year'] >= 2004) & (fatalities_df['Year'] <= 2015)]
      admissions_df = admissions_df[(admissions_df['Year'] >= 2004) & (admissions_df['Year'] <= 2015)]
```

This Code Explains:

```
# Fix Year columns
prescriptions_df['Year'] = prescriptions_df['Year'].str[:4].astype(int)
admissions_df['Year'] = admissions_df['Year'].str[:4].astype(int)
fatalities_df['Year'] = fatalities_df['Year'].astype(int)
```

- Sometimes the Year column contains extra characters like 2004-05 or has a string type (e.g., '2005\n').
- **.str[:4]:** This takes only the first 4 characters, which usually represent the actual year (e.g., '2005' from '2005-06').
- **.astype(int):** Converts the result to integer so that numerical filtering can be done.

Note: fatalities_df['Year'] is assumed to already be clean and just needs conversion to integer.

```
# Filter years
smokers_df = smokers_df[(smokers_df['Year'] >= 2004) & (smokers_df['Year'] <= 2015)]
prescriptions_df = prescriptions_df[(prescriptions_df['Year'] >= 2004) & (prescriptions_df['Year'] <= 2015)]
metrics_df = metrics_df[(metrics_df['Year'] >= 2004) & (metrics_df['Year'] <= 2015)]
fatalities_df = fatalities_df[(fatalities_df['Year'] >= 2004) & (fatalities_df['Year'] <= 2015)]
admissions_df = admissions_df[(admissions_df['Year'] >= 2004) & (admissions_df['Year'] <= 2015)]
```

- This filters out rows outside the year range 2004 to 2015.
- It ensures analysis is only on relevant data for the stated period.
- Example:
  - If a row has year 2002, it will be excluded.
  - If it has year 2008, it will be included.

5.6 Aggregate and Merge

```
# Convert values to numeric
fatalities_df['Value'] = pd.to_numeric(fatalities_df['Value'], errors='coerce')
admissions_df['Value'] = pd.to_numeric(admissions_df['Value'], errors='coerce')

# Aggregate smoking-related fatalities and admissions
# Smoking-related deaths
fatalities_clean = fatalities_df[fatalities_df['Diagnosis Type'].str.contains("caused by smoking", na=False)]
fatalities_clean = fatalities_clean.groupby('Year')['Value'].sum().reset_index().rename(columns={'Value': 'Smoking Related Deaths'})

# Smoking-related admissions
admissions_clean = admissions_df[admissions_df['Diagnosis Type'].str.contains("caused by smoking", na=False)]
admissions_clean = admissions_clean.groupby('Year')['Value'].sum().reset_index().rename(columns={'Value': 'Smoking Related Admissions'})


# Prepare key columns
# Extract relevant columns
smokers_filtered = smokers_df[['Year', '16 and Over']].rename(columns={'16 and Over': 'Smoking Rate (%)'})
prescriptions_filtered = prescriptions_df[['Year', 'All Pharmacotherapy Prescriptions']] \
    .rename(columns={'All Pharmacotherapy Prescriptions': 'Pharmacotherapy Prescriptions'})
metrics_filtered = metrics_df[['Year', 'Tobacco Price Index', 'Affordability of Tobacco Index']]

# Merge everything
final_df = smokers_filtered.merge(prescriptions_filtered, on='Year', how='outer') \
    .merge(metrics_filtered, on='Year', how='outer') \
    .merge(fatalities_clean, on='Year', how='outer') \
    .merge(admissions_clean, on='Year', how='outer') \
    .sort_values(by='Year').reset_index(drop=True)
```

This Code Explains:

```
# Convert values to numeric
fatalities_df['Value'] = pd.to_numeric(fatalities_df['Value'], errors='coerce')
admissions_df['Value'] = pd.to_numeric(admissions_df['Value'], errors='coerce')
```

- Ensures that 'Value' column is numeric (e.g., integer/float).
- If any value is invalid (e.g., text), it’s replaced with NaN due to errors='coerce'.

```
# Aggregate smoking-related fatalities and admissions
# Smoking-related deaths
fatalities_clean = fatalities_df[fatalities_df['Diagnosis Type'].str.contains("caused by smoking", na=False)]
fatalities_clean = fatalities_clean.groupby('Year')['Value'].sum().reset_index().rename(columns={'Value': 'Smoking Related Deaths'})

# Smoking-related admissions
admissions_clean = admissions_df[admissions_df['Diagnosis Type'].str.contains("caused by smoking", na=False)]
admissions_clean = admissions_clean.groupby('Year')['Value'].sum().reset_index().rename(columns={'Value': 'Smoking Related Admissions'})
```

- Keeps only the rows specifically related to smoking.
- **na=False:** Avoids error if Diagnosis Type is missing.
- Groups data by year, then sums the values to get total deaths and admissions.
- This simplifies the data to 1 row per year.

```
# Prepare key columns
smokers_filtered = smokers_df[['Year', '16 and Over']].rename(columns={'16 and Over': 'Smoking Rate (%)'})
prescriptions_filtered = prescriptions_df[['Year', 'All Pharmacotherapy Prescriptions']] \
    .rename(columns={'All Pharmacotherapy Prescriptions': 'Pharmacotherapy Prescriptions'})
metrics_filtered = metrics_df[['Year', 'Tobacco Price Index', 'Affordability of Tobacco Index']]
```

- Select only the necessary columns.
- Rename them to more readable or standard column names.

```
# Merge everything
final_df = smokers_filtered.merge(prescriptions_filtered, on='Year', how='outer') \
    .merge(metrics_filtered, on='Year', how='outer') \
    .merge(fatalities_clean, on='Year', how='outer') \
    .merge(admissions_clean, on='Year', how='outer') \
    .sort_values(by='Year').reset_index(drop=True)
```

- This creates a unified dataset with:
    - Smoking rates
    - Prescription data
    - Tobacco metrics
    - Smoking-related deaths
    - Smoking-related hospital admissions
- Uses outer join to keep all available data even if some values are missing in a given year.
- Sorted by Year.

Final Dataset:

```
final_df.head()
```

	Year	Smoking Rate (%)	Pharmacotherapy Prescriptions	Tobacco Price Index	Affordability of Tobacco Index	Smoking Related Deaths	Smoking Related Admissions
0	2004	25.0	2044.0	654.6	80.5	1336732.0	6511909.0
1	2004	26.0	2044.0	654.6	80.5	1336732.0	6511909.0
2	2004	23.0	2044.0	654.6	80.5	1336732.0	6511909.0
3	2005	24.0	2205.0	683.1	80.1	1310660.0	6635909.0
4	2005	25.0	2205.0	683.1	80.1	1310660.0	6635909.0

We've successfully combined the datasets into a unified table by `Year`. Here's what each column represents:

- **Smoking Rate (%)**: Proportion of people aged 16+ who smoke.
- **Pharmacotherapy Prescriptions**: Total cessation prescriptions.
- **Tobacco Price Index** & **Affordability Index**: Economic indicators.
- **Smoking Related Deaths** & **Admissions**: Aggregated health impacts.

5.7 Inspect Smoking Rate Data for Issues

```
print(final_df[['Year', 'Smoking Rate (%)']])
```

```
    Year  Smoking Rate (%)
0   2004              25.0
1   2004              26.0
2   2004              23.0
3   2005              24.0
4   2005              25.0
5   2005              23.0
6   2006              22.0
7   2006              23.0
8   2006              21.0
9   2007              21.0
10  2007              22.0
11  2007              20.0
12  2008              21.0
13  2008              22.0
14  2008              21.0
15  2009              21.0
16  2009              22.0
17  2009              20.0
18  2010              20.0
19  2010              21.0
20  2010              20.0
21  2011              20.0
22  2011              21.0
23  2011              19.0
24  2012              20.0
25  2012              22.0
26  2012              19.0
27  2013              19.0
28  2013              22.0
29  2013              17.0
30  2014              19.0
31  2014              20.0
32  2014              17.0
33  2015               NaN
```

- **Drop Duplicates + Missing Values**

```
final_df = final_df.dropna(subset=['Smoking Rate (%)', 'Smoking Related Deaths'])
final_df = final_df.drop_duplicates(subset=['Year'])
final_df = final_df.sort_values('Year').reset_index(drop=True)
```

- Remove rows with missing Smoking Rate (%) or Smoking Related Deaths.
- Eliminate duplicate years that caused line breaks.
- Sort data by Year, ensuring smooth plotting.

```
final_df
```

	Year	Smoking Rate (%)	Pharmacotherapy Prescriptions	Tobacco Price Index	Affordability of Tobacco Index	Smoking Related Deaths	Smoking Related Admissions
0	2004	25.0	2044.0	654.6	80.5	1336732.0	6511909.0
1	2005	24.0	2205.0	683.1	80.1	1310660.0	6635909.0
2	2006	22.0	2079.0	713.7	79.9	1254000.0	6597651.0
3	2007	21.0	2475.0	751.5	80.6	1233520.0	6635912.0
4	2008	21.0	2263.0	784.7	78.8	1238704.0	6894659.0
5	2009	21.0	2483.0	815.9	76.7	1187304.0	6897764.0
6	2010	20.0	2564.0	878.3	74.3	1170528.0	7056600.0
7	2011	20.0	2532.0	974.9	68.3	1110504.0	7169476.0
8	2012	20.0	2203.0	1057.8	66.2	1122872.0	7322506.0
9	2013	19.0	1778.0	1139.3	62.5	1123032.0	7446464.0
10	2014	19.0	1348.0	1226.0	59.4	1099180.0	7802520.0

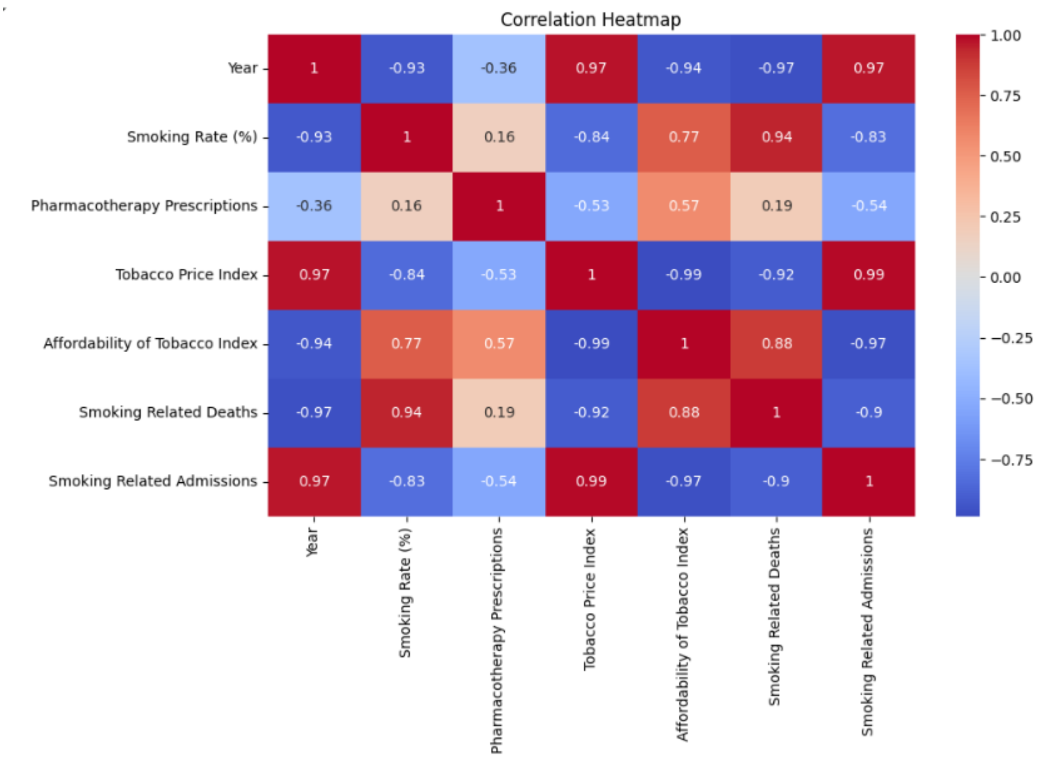
5.8 Exploratory Data Analysis (EDA)

```
plt.figure(figsize=(10, 6))
sns.heatmap(final_df.corr(numeric_only=True), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

This Code Explains:

- **final_df.corr()**:
  - Calculates the Pearson correlation matrix between all numerical columns.
  - Values range from **-1 to 1**:
    - **1**: perfect positive correlation
    - **-1**: perfect negative correlation
    - **0**: no correlation
- **sns.heatmap(..., annot=True, cmap="coolwarm")**:
  - Uses Seaborn to draw a colored heatmap of the correlations.
  - annot=True shows actual correlation values inside each box.
  - cmap="coolwarm" gives a blue-red color scheme:
    - Red = Positive correlation
    - Blue = Negative correlation
- **plt.title("Correlation Heatmap)**: Adds a title to the heatmap.

Output:



- **Year vs Tobacco Price Index (0.97)** and **Year vs Smoking Related Admissions (0.97)**:
  - As years progress, tobacco prices and smoking-related hospital admissions strongly increase.
- **Year vs Smoking Rate (-0.93)**:
  - As time progresses, smoking rate decreases – likely due to increased awareness, policy changes, or higher prices.
- **Tobacco Price Index vs Affordability of Tobacco (-0.99)**:
  - Very strong inverse correlation: as price increases, affordability decreases.
- **Smoking Rate vs Smoking Related Deaths (0.94)**:
  - A higher smoking rate correlates strongly with more smoking-related deaths.
- **Affordability vs Smoking Related Admissions (-0.97)**:
  - Less affordable tobacco is linked to fewer smoking-related admissions (possibly due to reduced consumption).
- **Pharmacotherapy Prescriptions**:
  - Has weak correlations with most variables (r ≈ 0.19 to 0.57), suggesting less direct influence.

```python
#Line Plot
import matplotlib.pyplot as plt

fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot smoking deaths (left Y axis)
ax1.plot(final_df['Year'], final_df['Smoking Related Deaths'], label='Smoking Deaths', color='tab:blue', marker='o')
ax1.set_xlabel('Year')
ax1.set_ylabel('Smoking Related Deaths', color='tab:blue')
ax1.tick_params(axis='y', labelcolor='tab:blue')

# Plot smoking rate (right Y axis)
ax2 = ax1.twinx()
ax2.plot(final_df['Year'], final_df['Smoking Rate (%)'], label='Smoking Rate (%)', color='tab:orange', marker='o')
ax2.set_ylabel('Smoking Rate (%)', color='tab:orange')
ax2.tick_params(axis='y', labelcolor='tab:orange')

plt.title("Smoking Rate vs Smoking Deaths")
fig.tight_layout()
plt.grid(True)
plt.show()
```

This Code Explains:

```python
import matplotlib.pyplot as plt

fig, ax1 = plt.subplots(figsize=(10, 6))
```

- Creates a figure and a primary axis (ax1) using matplotlib.
- figsize=(10, 6) sets the size of the plot (10 inches wide, 6 inches tall).

```python
# Plot smoking deaths (left Y axis)
ax1.plot(final_df['Year'], final_df['Smoking Related Deaths'], label='Smoking Deaths', color='tab:blue', marker='o')
```

- Plots years vs. smoking deaths using a blue line.
- marker='o' adds a dot at each data point for clarity.

```python
ax1.set_xlabel('Year')
ax1.set_ylabel('Smoking Related Deaths', color='tab:blue')
ax1.tick_params(axis='y', labelcolor='tab:blue')
```

- Labels the X-axis (Year) and the left Y-axis.
- Colors the Y-axis label and ticks in blue to match the line.

```python
# Plot smoking rate (right Y axis)
ax2 = ax1.twinx()
```

- Creates a second Y-axis (ax2) that shares the same X-axis as ax1.
- This is necessary because Smoking Rate (%) has a much smaller range than Smoking Deaths.

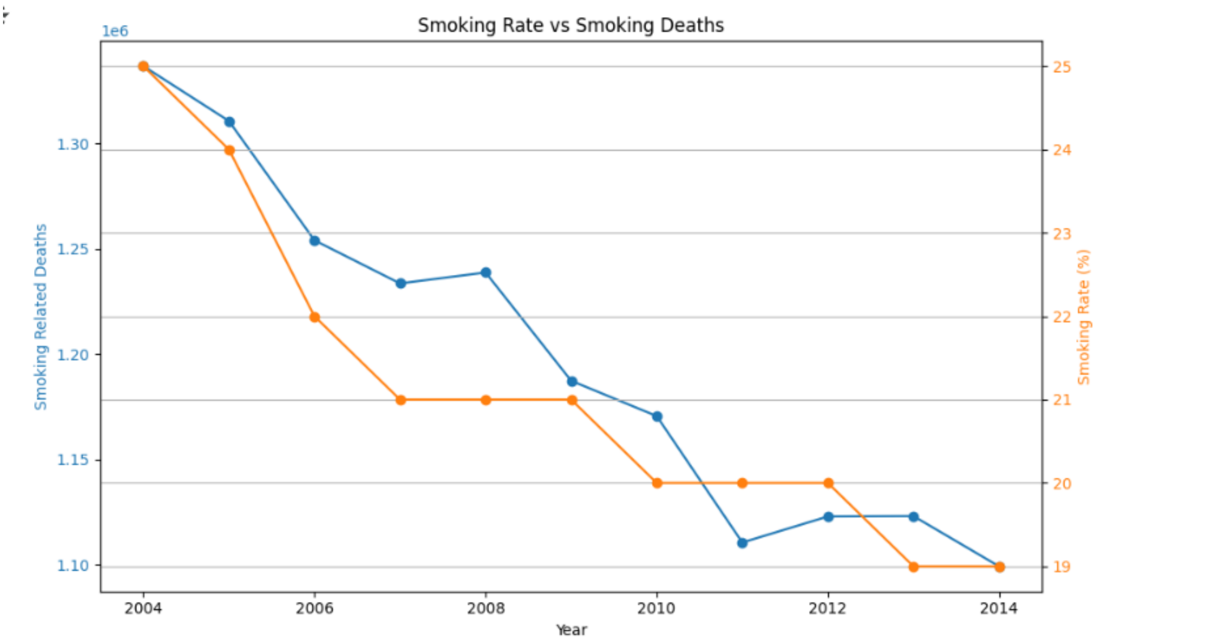
```python
ax2.plot(final_df['Year'], final_df['Smoking Rate (%)'], label='Smoking Rate (%)', color='tab:orange', marker='o')
ax2.set_ylabel('Smoking Rate (%)', color='tab:orange')
ax2.tick_params(axis='y', labelcolor='tab:orange')
```

- Plots the smoking rate with an orange line and circular markers.
- Sets Y-axis label and tick color to orange.

```python
plt.title("Smoking Rate vs Smoking Deaths")
fig.tight_layout()
plt.grid(True)
plt.show()
```

- Sets the plot title.
- **tight_layout()** adjusts spacing so labels/titles don't overlap.
- **grid(True)** adds grid lines for readability.
- **plt.show()** displays the final plot.

Output:



5.9 Machine Learning Model (e.g., Predict Smoking Deaths)

```python
# Define input and output
X = final_df[['Smoking Rate (%)', 'Pharmacotherapy Prescriptions', 'Tobacco Price Index', 'Affordability of Tobacco Index']]
y = final_df['Smoking Related Deaths']

# Drop missing
X = X.dropna()
y = y.loc[X.index]  # align with dropped rows

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("R² Score:", r2_score(y_test, y_pred))
```

```
R² Score: 0.9404984133937497
```

This Code Explains:

```
# Define input and output
X = final_df[['Smoking Rate (%)', 'Pharmacotherapy Prescriptions', 'Tobacco Price Index', 'Affordability of Tobacco Index']]
y = final_df['Smoking Related Deaths']
```

- X (features) includes 4 columns that might influence deaths:
  - Smoking Rate (%)
  - Pharmacotherapy Prescriptions – treatments to help quit
  - Tobacco Price Index – how prices have changed
  - Affordability Index – how easily people can still afford tobacco
- y is the target/output, i.e. the number of smoking-related deaths.

```
# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Splits the data:
  - 80% for training the model.
  - 20% for testing the model's accuracy.
- **random_state=42** ensures reproducibility (same split every time).

```
# Train model
model = LinearRegression()
model.fit(X_train, y_train)
```

- Creates a linear regression model object.
- Trains (fits) the model using the training data (X_train and y_train).
- The model learns how the input features relate to smoking-related deaths.

```
# Predict
y_pred = model.predict(X_test)
```

- Uses the trained model to predict deaths for the test input features (X_test).

```
# Evaluate
print("R² Score:", r2_score(y_test, y_pred))
```

- Calculates the R^2 (R-squared) Score — a common metric for regression:
  - 1.0= perfect prediction.
  - 0.0= model predicts no better than average.
  - Negative = worse than guessing.

Output: R^2 Score: 0.94:

- The model explains 94% of the variance in smoking-related deaths based on the input features:
  - Smoking Rate (%)
  - Pharmacotherapy Prescriptions
  - Tobacco Price Index
  - Affordability of Tobacco Index

- The relationship between your inputs and the number of smoking-related deaths is highly linear and predictable**.**

5.10 Save The Model

```
[317] joblib.dump(model, 'model.pkl')
```

```
['model.pkl']
```

6. Flask Web Application

Flask is a lightweight web framework in Python used to host the model.

6.1 Objective:

To develop a lightweight, interactive web interface for visualizing and predicting tobacco-related health metrics using the cleaned and analyzed dataset.

7. Steps in Flask App Development:

7.1 Project Structure

- Tobacco Use And Mortality, 2004-2015
    - app.py
    - model.pkl
    - templates
        - index.html

7.2 Flask Backend Logic (app.py)

app.py > ...

```python
from flask import Flask, render_template, request
import joblib

app = Flask(__name__)
model = joblib.load("model.pkl")  # Make sure this file exists in the same directory

@app.route("/")
def home():
    return render_template("index.html", prediction=None)

@app.route("/predict", methods=["POST"])
def predict():
    try:
        inputs = [
            float(request.form["smoking_rate"]),
            float(request.form["pharma_presc"]),          # FIXED name
            float(request.form["price_index"]),
            float(request.form["afford_index"]),          # FIXED name
        ]
        prediction = model.predict([inputs])[0]
        prediction = round(prediction, 2)
    except Exception as e:
        prediction = f"Error: {e}"

    return render_template("index.html", prediction=prediction, form=request.form)

if __name__ == "__main__":
    app.run(debug=True)
```

This Code Explains:

.py > ...

```python
from flask import Flask, render_template, request
import joblib
```

- **Flask**: Used to create the web server.
- **render_template**: Loads and displays an HTML file (index.html).
- **request**: Gets user input from the form on the web page.
- **joblib**: Used to load the pre-trained ML model (model.pkl).

```python
app = Flask(__name__)
model = joblib.load("model.pkl")  # Make sure this file exists in the same directory
```

- **Flask(__name__)**: Initializes the Flask app.
- **joblib.load**: Loads trained machine learning model from the file model.pkl.

```python
@app.route("/")
def home():
    return render_template("index.html", prediction=None)
```

- **Route /**: Displays the main page with a form.
- It initially loads index.html with no prediction (just the form).

```
@app.route("/predict", methods=["POST"])
def predict():
```

- **Route /predict**: This handles the form submission.
- It accepts POST method (used when user submits the form).

```
    try:
        inputs = [
            float(request.form["smoking_rate"]),
            float(request.form["pharma_presc"]),          # FIXED name
            float(request.form["price_index"]),
            float(request.form["afford_index"]),          # FIXED name
        ]
```

- **request.form[...]**: Fetches values from the form fields.
- Converts them into float and stores them in a list called inputs.
- These match the expected features used when training the model.

```
    prediction = model.predict([inputs])[0]
    prediction = round(prediction, 2)
```

- **model.predict(...)**: Uses the ML model to make a prediction with the input values.
- **[0]**: Gets the actual predicted value from the list/array.
- **round(...)**: Rounds the result to 2 decimal places.

```
except Exception as e:
    prediction = f"Error: {e}"
```

- If there's an error (e.g., input can't be converted to float), it catches the exception and displays an error message.

```
return render_template("index.html", prediction=prediction, form=request.form)
```

- Renders index.html again, now passing the prediction and form values back to the page.
- This allows the user to see the result and keep their inputs in the form.

```
if __name__ == "__main__":
    app.run(debug=True)
```

- **debug=True**: Enables live reloading and shows errors in the browser for easier development.

7.3 User Interface (templates/index.html)

This is the frontend HTML page for Flask web app. It lets users input data to predict smoking-related deaths using ML model.

Main Sections Explained:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Smoking-Related Deaths Predictor</title>
    <style>
        body {
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
            background: linear-gradient(135deg, #f6d365 0%, #fda085 100%);
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
        }

        .container {
            background: rgba(255, 255, 255, 0.95);
            backdrop-filter: blur(10px);
            border-radius: 20px;
            padding: 40px;
            max-width: 500px;
            width: 90%;
```

```html
    </style>
</head>
<body>
    <div class="container">
        <h2>🚬 Smoking-Related Deaths Predictor</h2>
        <form method="POST" action="/predict">
            <label for="smoking_rate">Smoking Rate (%)</label>
            <input type="number" step="0.1" min="0" name="smoking_rate" id="smoking_rate"
                required value="{{ request.form.smoking_rate or '' }}">

            <label for="pharma_presc">Pharmacotherapy Prescriptions</label>
            <input type="number" step="0.1" min="0" name="pharma_presc" id="pharma_presc"
                required value="{{ request.form.pharma_presc or '' }}">

            <label for="price_index">Tobacco Price Index</label>
            <input type="number" step="0.1" min="0" name="price_index" id="price_index"
                required value="{{ request.form.price_index or '' }}">

            <label for="afford_index">Affordability of Tobacco Index</label>
            <input type="number" step="0.1" min="0" name="afford_index" id="afford_index"
                required value="{{ request.form.afford_index or '' }}">

            <button type="submit">Predict</button>
        </form>

        {% if prediction is not none %}
            <div class="result">
                Predicted Smoking-Related Deaths: {{ prediction }}
            </div>
        {% endif %}
    </div>
</body>
</html>
```

This Code Explains:

1. HTML Structure

```html
<!DOCTYPE html>
<html lang="en">
```

- Declares the document as HTML5.
- Sets the language to English.

2. Head Section

- Sets metadata, title, internal CSS styling.
- style tag:
    - The whole body: centered layout with a gradient background.
    - The form container: white card-like box with shadows and blur.
    - Input fields: number-only with step 0.1, min 0, rounded borders.
    - Button: colorful gradient, hover effects.
    - Result display: a styled div shown after prediction.

3. Body Section

- Contains the visual content of the page.
- **<form>:**

    - Submits data via POST to /predict (handled by Flask).
    - Contains 4 input fields:
        - smoking_rate
        - pharma_presc
        - price_index
        - afford_index
- **All inputs:**
    - Must be ≥ 0
    - Accept decimal values (via step="0.1")
    - Use oninput="enforceMinValue(event)" to prevent invalid input
- **<button>:**
    - Submits the form to the backend for prediction.
- **{% if prediction is not none %}:**

```
{% if prediction is not none %}
    <div class="result">
        Predicted Smoking-Related Deaths: {{ prediction }}
    </div>
{% endif %}
```

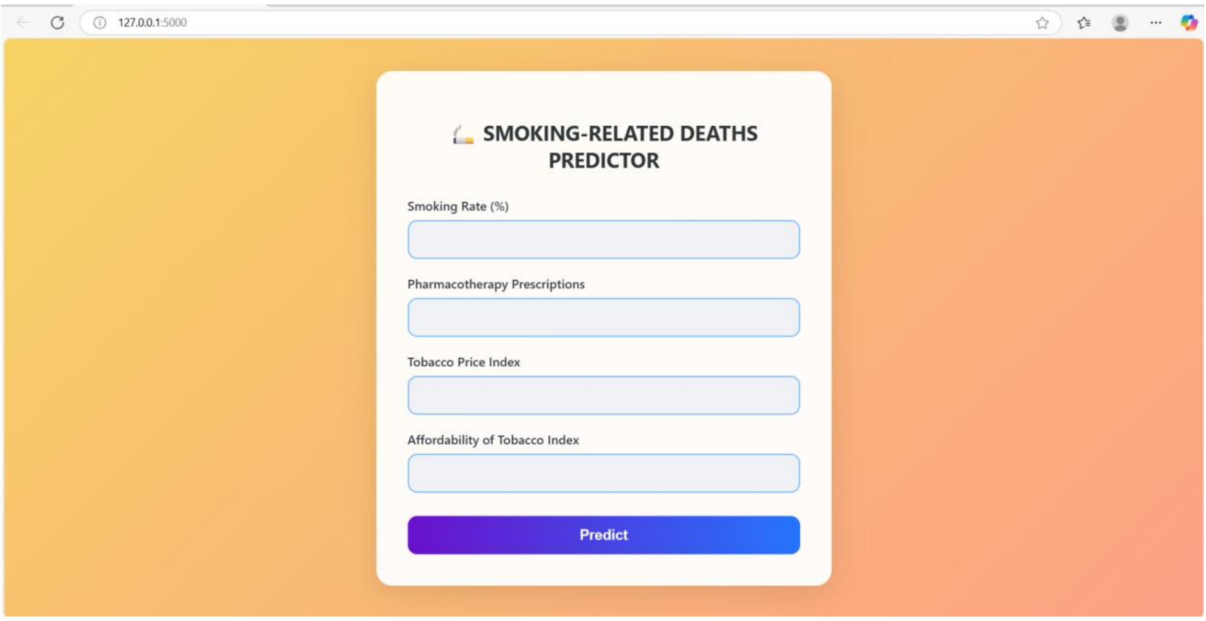
    - Uses Flask/Jinja2 templating to conditionally show the prediction result.

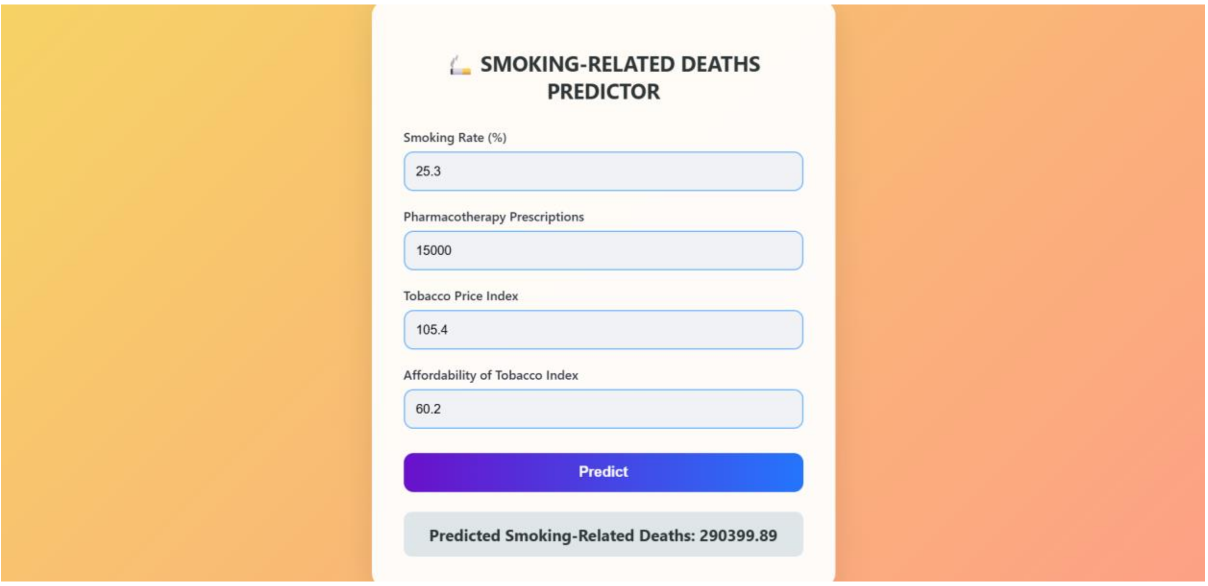
7.4 Running the App

```
PS D:\VS code projects\Tobacco Use And Mortality, 2004-2015> .venv\Scripts\activate
(.venv) PS D:\VS code projects\Tobacco Use And Mortality, 2004-2015> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

The app runs on http://127.0.0.1:5000.

Output:





Observations:

- **Pharmacotherapy prescriptions** could be 0 (some countries may have no prescriptions).
- **Affordability Index** or **Price Index** may be below 1 depending on how index is scaled.
- **Smoking Rate** could be below 1% in rare cases.

8. Key Insights

- **Strong Correlation Between Smoking and Mortality:**
    - Smoking Rate and Smoking-Related Deaths show a high positive correlation (r ≈ 0.94), indicating that increased smoking prevalence directly contributes to more deaths.

- **Decline in Smoking Rates Over Time:**
    - Smoking rates have decreased significantly from 2004 to 2015, correlating negatively with time ($r \approx -0.93$), likely due to public health initiatives, policy changes, and price increases.
- **Impact of Economic Factors:**
    - Tobacco Price Index increased over the years, and as prices rose, tobacco became less affordable (affordability index correlation ≈ -0.99).
    - There is a strong inverse correlation between affordability and smoking-related admissions ($r \approx -0.97$) — indicating reduced affordability may discourage use and result in fewer hospital admissions.
- **Effectiveness of Cessation Support (Pharmacotherapy):**
    - Pharmacotherapy prescriptions showed weak correlations ($r \approx 0.19$–0.57) with health outcomes, suggesting these interventions may have limited standalone impact or may be underutilized.
- **Machine Learning Model Performance:**
    - A Linear Regression model trained on smoking and economic indicators achieved an R^2 score of 0.94, meaning it can predict tobacco-related deaths with high accuracy.
- **Web Deployment:**
    - The model was successfully integrated into a Flask-based web application, enabling real-time predictions of smoking-related deaths based on user inputs.

9. Conclusion

This project demonstrates a robust relationship between tobacco use and mortality in the U.S. from 2004 to 2015. The findings validate that policy-driven increases in tobacco pricing and reduced affordability are effective in curbing smoking rates and associated deaths. While cessation support (like prescriptions) plays a role, its influence appears secondary compared to economic deterrents. The predictive model further offers a valuable tool for policymakers and health professionals to forecast health outcomes and guide intervention strategies. The successful deployment of this model in a web app makes these insights accessible and actionable for broader use.